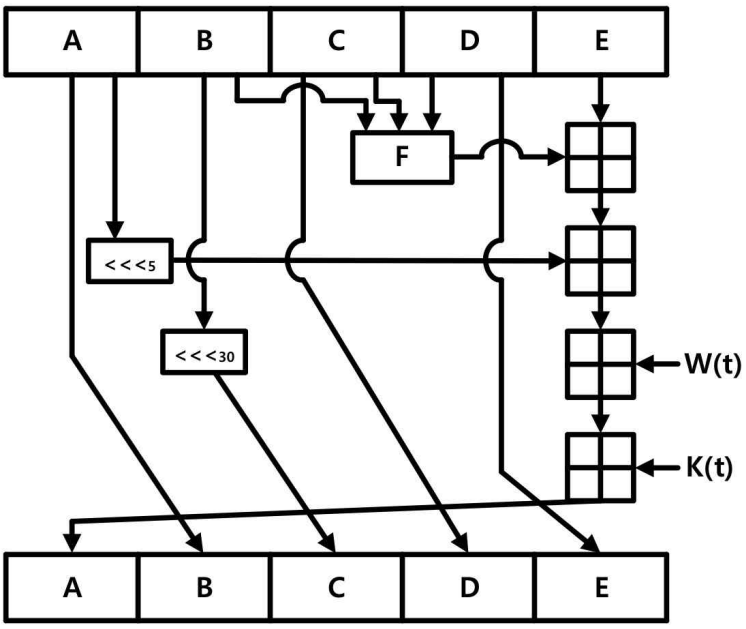
<project. SHA-1 Hash Processor Design>

2020112099 송민지

SHA-1이란 임의의 길이의 입력 데이터를 160비트의 출력 데이터로 바꾸는 것을 의미한다. 이번 과제에서 우리는 512 비트 블록의 입력을 받아 160 비트를 출력하고자 한다. 입력된 512비트를 32비트 word로 다섯 조각으로 나눠 앞에서부터 차례로 A, B, C, D, E로 이름을 붙인다. 이들은 아래 그림과 같이 circular left shift 연산자, W(t), K(t), F(t,b,c,d)의 과정을 거쳐서 암호화된 값을 출력한다.



코드를 구현할 때는 베릴로그 코드로 작성하기 전에 c 코드로 작성한 코드를 보고 이를 바탕으로 진행하고자 하였다. 우선 A, B, C, D, E는 각각 32비트의 초기값을 갖고 있는데, 이를 <그림 1.1>처럼 각자 parameter로 받아와서 초기값으로 고정해두었다.

```
parameter a=32'h67452301;
parameter b=32'hEFCDAB89;
parameter c=32'h98BADCFE;
parameter d=32'h10325476;
parameter e=32'hC3D2E1F0;
```

<그림 1.1>

이렇게 초기값을 설정해둔 후, 구현에 필요한 것들을 function으로 각각 구현했는데, 위의 암호화 과정을 진행하는 것을 decode function에 구현하였다. 구현 방식은 아래의 <그림 1.2>와 같은데, A, B, C, D, E를 각각 32비트의 a,b,c,d,e로 입력을 받고, t가 0부터 80까지 흘러가는 동안 실행하고자 한다. 만약 e에는 d 값을 넣고, d에는 e 값을 넣고, c에는 b를 <<<30한 값을 넣고, b에는 a 값을 넣고, a에는 위 과정이 진행되기 전 원래 값들을 이용하여 e와 F(t,b,c,d)와 a를 <<<5한 값과 k(t)를 더한 값을 넣어준다. 이를 구현할 때는 case를 이용하여 나눠준다.

```
function decode;
input [6:0] t;
input [31:0] a,b,c,d,e;

reg [31:0] temp;
reg [31:0] prestate;

begin
    for(t=0;t<80;t=t+1)
        begin
            temp= e + F(t,b,c,d) + circular_leftshift(a,5) + w(t) + k(t);
            case(prestate)
            e:
                e=d;
            d:
                d=c;
            c:
                c=circular_leftshift(b,30);
            b:
                b=a;
            a:
                a=temp;
            endcase
        end
    end
endfunction
```

<그림 1.2>

<<<라는 연산자를 수행하기 위한 코드를 circular_leftshift에 구현하였다. 이 연산자는 왼쪽으로 이동하고자 할 때 맨 앞자리의 수를 잘라내는 것이 아니라 제일 뒤로 옮겨서 rotate 하도록 만들어주는 역할을 한다. 이 연산자에는 32비트의 값이 들어가므로 32비트로 지정을 하였으며, num 값은 32 이하이므로 5비트로 지정해 주었다. <그림 1.3>은 이러한 circular_leftshift 함수의 구현 코드이다.

```
function [31:0] cirular_leftshift;
input [31:0] temp;
input [4:0] num;

begin
    cirular_leftshift= (temp<<num) | (temp>>(32-num));
end
endfunction
```

<그림 1.3>

a를 암호화하는 과정에서 필요한 k(t)를 위하여 k 함수를 구현하였다. <그림 1.4>는 k 함수인데, k에서는 t의 값에 따라 k의 값이 달라지는 것을 구현하고자 하였다. 각자 32비트 값을 반환하므로 32비트로 선언을 해주었으며 t 값은 위에서와 마찬가지로 80을 넘지 않으므로 7비트로 구현하였다. 0<t<20, 20<t<40, 40<t<60, 60<t<80으로 나눠서 k의 값을 각각 넣어주었다.

```
function [31:0] k;
input [6:0] t;
begin
    if (t >= 0 && t < 20)
        k=32'h5A827999;
    else if (t >= 20 && t < 40)
        k=32'h6ED9EBA1;
    else if (t >= 40 && t < 60)
        k=32'h8F1BBCDC;
    else if (t >= 60 && t < 80)
        k=32'hCA62C1D6;
end
endfunction
```

<그림 1.4>

a를 암호화하는 과정에서 필요한 F(t,b,c,d)를 위하여 F 함수를 구현하였다. F에서는 input 값으로 t, b, c, d를 받아오도록 설정하고 32비트의 값을 입력 받으므로 32비트로 구현하고자 하였다. 또 t 값은 위에서와 마찬가지로 80을 넘지 않으므로 7비트로 구현하였다. 0<t<20, 20<t<40, 40<t<60, 60<t<80으로 나눠서 범위에 따른 F의 연산자를 구현하고자 하였다. 구현 방식은 아래의 <그림 1.5>와 같다.

```
function [31:0] F;
input[6:0] t;//
input [31:0] b,c,d;

begin
    if (t >= 0 && t < 20)
        F = (b & c) | ((~b) & d);
    else if (t >= 20 && t < 40)
        F = (b ^ c ^ d);
    else if (t >= 40 && t < 60)
        F = ((b & c) | (b & d) | (c & d));
    else if (t >= 60 && t < 80)
        F = (b ^ c ^ d);
end
endfunction
```

<그림 1.5>